Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Design Patterns and SOLID with C

Class Duration

21 hours of live training delivered over 3 days.

Student Prerequisites

- Professional experience writing C#
- Comfort with classes, interfaces, and generics
- Familiarity with ASP.NET Core is helpful but not required
- No prior design patterns knowledge required

Target Audience

This course is designed for working C# developers who want to make better design decisions on the modern .NET platform. It suits engineers maintaining growing .NET codebases, developers who learned patterns from older Java-flavored material and want a current treatment, and team leads who review code — including code produced by AI assistants — and need a shared vocabulary for evaluating structure. Participants should write C# professionally; no prior patterns background is assumed.

Description

C# has absorbed many of the Gang of Four patterns into the language and the platform: delegates and events replaced hand-rolled Observer, LINQ is a pipeline pattern in the standard library, and ASP.NET Core ships a dependency injection container in the box. This course teaches SOLID and the GoF catalog the way they should be taught on .NET 10 with C# 14: selectively and honestly, covering the roughly fifteen patterns that still earn their keep in contemporary code — split across creational, structural, and behavioral families — and showing where a language feature has made the classic implementation obsolete.

The course is built around refactoring rather than recitation. Each pattern is motivated by a real code smell, introduced as the refactoring that resolves it, and weighed against simpler alternatives so participants learn when a pattern is overengineering. Modern C# gets equal billing with the classics: interfaces with default members, generics and variance, delegates and lambdas versus behavioral patterns, records and immutability, pattern

matching versus Visitor, `IDisposable` and async resource patterns, and nullable reference types as a design constraint. The built-in DI container is treated as a first-class topic, including lifetimes and the seams it creates for testing. The final day addresses design in the age of AI-generated code: recognizing pattern use (and misuse) in agent output, reviewing generated designs against SOLID, and steering coding assistants toward structures your team can maintain.

Learning Outcomes

- Apply each SOLID principle to real C# code and recognize when a principle is being over-applied
- Identify the design problem each major GoF pattern solves and select among creational, structural, and behavioral options
- Use delegates, events, and lambdas in place of hand-rolled Strategy, Command, and Observer implementations
- Design with interfaces effectively, including default interface members, generics, and variance
- Apply records, init-only setters, and immutability as structural design tools
- Replace Visitor-style double dispatch and conditional sprawl with pattern matching where it is clearer
- Use the built-in .NET dependency injection container with appropriate lifetimes, and recognize captive-dependency bugs
- Apply `IDisposable`, `IAsyncDisposable`, and async patterns to resource management designs
- Recognize LINQ as the platform’s pipeline pattern and design composable pipelines deliberately
- Use nullable reference types to make optionality explicit in API and object design
- Refactor toward patterns incrementally from concrete code smells, and recognize anti-patterns and overengineering
- Review AI-generated C# for structural quality, naming the patterns it uses and the SOLID violations it hides

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Software Requirements

- .NET 10 SDK
- Visual Studio 2026 or VS Code with the C# Dev Kit
- Git for version control
- Access to an AI coding assistant for the code review exercises (any major tool)

Training Topics

Design Principles Foundations

- What design patterns are and what they are not
- Coupling, cohesion, and the cost of change
- Reading the GoF catalog critically in 2026
- How C# and .NET have absorbed the catalog
- The vocabulary value of patterns on a team

SOLID Principles, Applied Honestly

- Single Responsibility: reasons to change, not line counts
- Open/Closed via interfaces, delegates, and extension
- Liskov Substitution, inheritance, and sealed types
- Interface Segregation and role interfaces
- Dependency Inversion and the composition root
- When SOLID is over-applied: speculative abstraction

Interfaces, Generics, and Types in Modern C

- Interface design and default interface members
- Generics in pattern implementations
- Covariance and contravariance in practice
- Records, init-only setters, and immutability
- Nullable reference types as a design constraint
- Pattern matching and exhaustive switches

Creational Patterns

- Factory Method and Abstract Factory: when they pay off
- Static factory methods and named constructors
- Builder vs. object initializers, records, and `with`
- Singleton vs. container-managed lifetimes
- Options pattern for configuration objects

Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Structural Patterns

- Adapter for integrating third-party code
- Facade for taming complex subsystems
- Decorator and pipeline composition with DI
- Proxy, lazy initialization, and `Lazy<T>`
- Composite for tree structures
- Middleware as a structural idiom

Behavioral Patterns

- Strategy via interfaces vs. delegates and lambdas
- Command, `Func/Action`, and undo stacks
- Observer vs. events and `IObservable<T>`
- Template Method vs. composition and hooks
- State machines: enums, classes, and switches
- Visitor vs. pattern matching on type hierarchies
- Iterator, `IEnumerable<T>`, and `IAsyncEnumerable<T>`

LINQ, Async, and Resource Patterns

- LINQ as the pipeline pattern in the standard library
- Deferred execution and composability
- `IDisposable`, `IAsyncDisposable`, and `using` declarations
- Async patterns: `Task`, `ValueTask`, and cancellation
- Designing APIs that compose under async

Dependency Injection on .NET

- The built-in container: registration and resolution
- Service lifetimes: singleton, scoped, transient
- Captive dependencies and other lifetime bugs
- Keyed services and decorating registrations
- Testability as a design feedback signal

Refactoring, Anti-Patterns, and Overengineering

- Code smells that motivate each pattern family
- Incremental refactoring with tests as a safety net
- Case study: replacing conditional sprawl with Strategy
- Pattern abuse: ceremony inherited from older .NET
- Premature abstraction, YAGNI, and the rule of three

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Patterns in the Age of AI-Generated Code

- Structural habits of AI coding assistants
- Recognizing pattern use and misuse in agent output
- Reviewing generated code against SOLID
- Steering agents with design-aware prompts and conventions
- Keeping architectural ownership when agents write the code